Computer Assisted Methods in Engineering and Science, **24**: 89–100, 2017.

Entropy-based regularization of AdaBoost

Michał Bereta
Institute of Computer Science
Cracow University of Technology
Warszawska 24, 31-155 Kraków, Poland
e-mail: mbereta@pk.edu.pl

In this study, we introduce an entropy-based method to regularize the AdaBoost algorithm. The AdaBoost algorithm is a well-known algorithm used to create aggregated classifiers. In many real-world classification problems in addition to paying special attention classification accuracy of the final classifier, great focus is placed on tuning the number of the so-called weak learners, which are aggregated by the final (strong) classifier. The proposed method is able to improve the AdaBoost algorithm in terms of both criteria. While many approaches to the regularization of boosting algorithms can be complicated, the proposed method is straightforward and easy to implement. We compare the results of the proposed method (EntropyAdaBoost) with the original AdaBoost and also with its regularized version, ϵ-AdaBoost on several classification problems. It is shown that the proposed methods of EntropyAdaBoost and ϵ-AdaBoost are strongly complementary when the improvement of AdaBoost is considered.

Keywords: AdaBoost, regularization, entropy, EntropyAdaBoost.

1. Introduction

Among many approaches to classification problems, such as neural networks (NN) or support vector machines (SVM) [1], the boosting family of algorithms takes a noticeable place. In general, the boosting approach in classification problems is designed to create a strong classifier as the aggregation of the so-called weak (base) classifiers or learners. A weak classifier is usually considered to perform only slightly better than a random guess about the class label of an object being classified. In boosting, training examples are associated with weight distribution. Based on the current weight distribution, each subsequent weak classifier is trained, focusing the most on examples with higher weights, which reflects the relative difficulty of these examples for the previous weak learners. Upon concluding training of the current weak classifier, this classifier is added to the set of aggregated classifiers with the weight depending on its classification performance, and the weights of the examples are modified. The final classification decision is made based on the weighted majority voting of all base classifiers.

AdaBoost is the basic boosting algorithm for two-class classification problems [2, 3] and it is the basis for developing other boosting algorithms. Despite its relative simplicity, it can be successfully applied to difficult classification problems. One noteworthy example is the Viola-Jones face detection framework which is, in fact, a standard framework nowadays [4]. One of the sources of its success is its ability to efficiently process a huge number of features. In this framework, as in many other applications of boosting, the base classifiers are as simple as when operating on just one feature. This kind of classifier is called a stump classifier or a decision stump and can be considered as an extremely simple decision tree, with just one node (the root node) performing only one splitting test on the selected feature. In this paper, we also consider decision stumps as the base classifiers.

From the description of boosting given above, one can conclude that it is desirable to have the final strong classifier consisting of as small as possible number of base learners, without sacrificing the classification accuracy. A smaller number of decision stumps means that the strong classifier is more efficient and faster, which is important in some applications. Additionally, a lower number of features have to be evaluated to make the final decision, which is important in some applications in which features are computed on demand. The Viola and Jones framework for face detection is one example of such approach.

To summarize, when training a boosting classifier we care about the classification accuracy of a final strong classifier and (in some applications) about the number of weak classifiers that are aggregated. The optimal number of weak classifiers that should be aggregated is not known and is usually selected empirically, for example, by cross-validation. While this allows for checking what is the best performing number of weak learners, additional methods are needed if we are interested in minimizing it. Regularization methods can be used for this purpose.

1.1. Regularization

Regularization can be defined as any method affecting the original learning algorithm, based on the properties of the training data or the specific characteristics of a classifier being trained. It can be used to prevent overfitting the training data which would result in poor generalization ability of a classifier. For some time, it was considered that the regularization is not necessary in the case of boosting due to its great generalization ability [5]. For example, when using a neural network as a classifier, adding too many neurons and training the network for too many generations, can result in an over-trained network with poor generalization. In the case of boosting, a similar effect can be expected when subsequent weak learners are added. In fact, boosting is very resistant to overfitting in such situations. Nevertheless, it is not entirely free from this fault. Regularizing boosting by adjusting the procedure of training the weak learners can be considered as a regularization procedure. While it is believed [6] that regularization is only seldom necessary for boosting when the classification performance is considered, it can be also useful when simpler classifiers, i.e., with a smaller number of base learners are preferred.

1.2. Motivation of this study

In this work, we propose a novel method of AdaBoost regularization. The goal is to create a simpler final strong classifier, without sacrificing the classification accuracy. The proposed method is based on a modification of the criteria used to select subsequent weak learners. In the original AdaBoost, this is done based on minimization of the weighted error of the weak learner. In the proposed modification, an additional term is added to the weighted classification error. This term is based on the entropy measure of the selection rate of features selected by previous weak learners. We enforce the boosting procedure to prefer a set of base learners that collectively select the features more uniformly, i.e., with a bigger entropy. Furthermore, in the numerical tests it is presented that this modification can have a positive effect on the final classifier, both in terms of the classification accuracy and the number of the weak learners needed to solve the problem.

The motivation of this approach comes from the known regularization methods for NN training. In the case of NN, several regularization methods work as a way to limit the absolute value of the weight associated with a given input signal to the neuron. If the weight is allowed to be arbitrarily large (given its absolute value), the inner state of the neuron, and thus its output signal, depend primarily on the value of the feature with the strongest weight. This can have a detrimental effect on the ability of the NN to discover the usefulness of other features (input signals in general) and the relationships among them. Regularization for NN often implements a forgetting effect, i.e., the value of the weight decreases over time, unless it is being further stimulated by training examples. The proposed method of AdaBoost regularization is an attempt to introduce similar mechanism

to the boosting framework, and it is especially easy to implement for decision stumps used as base learners. In this paper, we show that by adding a term measuring the entropy of selection rate of features to the main criteria (weighted error), we can regularize the boosting procedure.

One possible explanation of the successes of boosting is that while each weak classifier is selected based on the minimized weighted error, the resulting strong classifier is minimizing the exponential loss. Boosting can also be formulated as a stage-wise functional gradient descent [7]. In this approach, the simple original formulation of AdaBoost is converted in to somewhat more complicated form. However, this opens up the possibilities to use arbitrarily loss functions, which can be modified with different regularization terms. Most work on the boosting regularization focuses on this approach [8]. Many such algorithms are much more complicated than the original boosting algorithm. For example, fully corrective boosting algorithms, for example TotalBoost or LPBoost [9], require that the weights of the previously learned weak classifiers are updated in each boosting iteration. Often, this introduces additional optimization problem that has to be solved in each iteration of boosting. Thus, the computational cost increases. In contrast to the aforementioned approaches, the method proposed in this paper does not modify significantly the original simple formulation of AdaBoost. Thus, it is easy to implement.

To the best of author's knowledge, the proposed approach to regularize AdaBoost with the usage of entropy term is first such an attempt presented in literature. Obviously, entropy has been used in many machine learning algorithms. The best example of which is the fact that most algorithms for growing decision trees use entropy-like criterions to select the best training data split in each node. In the case of boosting an algorithm Ent-Boost was proposed in [10], which employs entropy to estimate the optimal number of bins used to performed the discretization of features' values. However, this problem is different from the one discussed in this paper.

1.3. Contributions of this work

There are several contributions of this article. First, we propose a straightforward way to regularize the boosting framework for decision stumps being used as weak learners. We compare the proposed method with the original AdaBoost and with an other popular regularization method called ϵ-AdaBoost. The results of numerical experiments on several classification problems are analyzed in a statistically rigorous way. We show that while none of the two regularization methods considered are universal, they are highly complementary. This means that in each considered classification problem it is possible to improve AdaBoost by using at least one of the two methods. On the other hand, neither ϵ-AdaBoost nor the proposed methods are capable to outperform the AdaBoost features on its own. As a result, the method proposed in this work, together with the traditional ϵ-AdaBoost, form a strong tool to tune the boosting classifiers for cases when the number of weak learners should be minimized without sacrificing the classification accuracy.

The rest of this paper is organized as follows. In Sec. 2 we review the original AdaBoost algorithm. In Sec. 3, we introduce the proposed entropy-based modification of AdaBoost. In Sec. 4, we compare the performance of the proposed method with that of the original AdaBoost and its regularized version, ϵ-AdaBoost, on selected classification problems. The results are analyzed by the Friedman statistical test with a set of post-hoc comparisons. The conclusions are drawn in Sec. 5.

2. Adaboost overview

In this section, we review the discrete AdaBoost algorithm presented in [11, 12]. Despite being one of the simplest boosting algorithms, it was successfully applied to many real-world problems. One example is the face detection framework of Viola and Jones [4]. Here we remind AdaBoost as Algorithm 1. In general, the final strong classifier is constructed step by step, by learning a new weak classifier in each iteration based on the current weight distribution of the training examples. The currently trained weak learner focuses more on the training examples that were problematic

for the previous base learners. Each m-th weak learner is given a weight α_m (Eq. (2)), which reflects its classification performance during training. The final answer is given as the weighted majority voting (Eq. (5)).

Input: Training data $(\mathbf{x}_i, y_i)$, $y_i \in \{-1, 1\}$, $x_i \in \mathbb{R}^K$, K – the number of features, $i = 1, ..., N$; M – the number of the weak learners to learn.
Output: $H(\mathbf{x})$ – the aggregated classifier
1. Initialize the weight distribution of training examples as $w_i = 1/N$.
2. **for** *m=1 to M* **do**
   a. Train a weak classifier $h_m(\mathbf{x})$ using the current weight distribution.
   b. Calculate the weighted error of $h_m(\mathbf{x})$ as

$$error_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq h_m(\mathbf{x}_i))}{\sum_{i=1}^{N} w_i} \tag{1}$$

   c. Calculate the weight α_m of h_m as

$$\alpha_m = \log\left(\frac{1 - error_m}{error_m}\right) \tag{2}$$

   d. Update the weight distribution as

$$w_i = w_i e^{\alpha_m I(y_i \neq h_m(\mathbf{x}_i))} \tag{3}$$

$$w_i = \frac{w_i}{\sum_{j=1}^{N} w_j} \tag{4}$$

end
3. **Return** the aggregated classifier as

$$H(\mathbf{x}) = sign\left(\sum_{i=1}^{M} \alpha_m h_m(\mathbf{x})\right) \tag{5}$$

Algorithm 1: AdaBoost algorithm.

AdaBoost is formulated as Algorithm 1 for any type of weak learner. However, boosting usually works best when the base learners are simple, and not much better than a random guess. The usual choice for the weak learner is the decision tree. In this work, we use a decision stump, which is an extremely simple decision tree, with only one node (the root node) in which only one selected feature is tested. Next, the threshold test is performed, which compares the value of the selected feature with a threshold value selected during training. Usually, the threshold value is selected as the one minimizing the weighted error on the training examples. The usage of the decision stump is especially convenient for the regularization approach proposed in this paper, as it is very straightforward to calculate the entropy of the selection rate of the features. The detailed algorithm to create a decision stump is given as Algorithm 2.

In this work, we also use ϵ-AdaBoost as another simple method of boosting regularization. In ϵ-AdaBoost, each weak classifier has the constant weight α_m, which does not depend on the

classification accuracy of the weak learner during training as it is written in the original AdaBoost. The constant weight ϵ is a hyperparameter of the algorithm. This is usually a small value selected empirically by a method such as cross-validation. The only difference compared to the original AdaBoost is in step 2c of Algorithm 1, where α_m is always set to ϵ. In [13], ϵ-AdaBoost was proved to asymptotically converge to the l_1-norm regularized version of AdaBoost, and as such it will be used in this work for comparison.

Input: Training data $(\mathbf{x}_i, y_i)$, $y_i \in \{-1, 1\}$, $x_i \in \mathbb{R}^K$, K – the number of features, $i = 1, ..., N$; $\{w_i\}$ – weight distribution of the training examples.
Output: A decision stump $h(\mathbf{x})$ minimazing the weighted classification error;
$h(\mathbf{x}) = \{j, threshold, class\_id\}$, where $class\_id \in \{-1, 1\}$, $j \in \{1, ..., K\}$, $threshold \in \mathbb{R}$
1. Find j, $threshold$, and $class\_id$, so that for $h(\mathbf{x}) = \{j, threshold, class\_id\}$, the weighted classification error

$$error(h(\mathbf{x})) = \sum_{i=1}^{N} w_i I(h(\mathbf{x}_i \neq y_i)) \tag{6}$$

is minimized.
2. **Return** $h(\mathbf{x}) = \{j, threshold, class\_id\}$ as

$$h(\mathbf{x}) = \begin{cases} class\_id & \text{if } x_j \geq threshold \\ -1 \cdot class\_id & \text{if } x_j < threshold \end{cases} \tag{7}$$

Algorithm 2: Decision stump learning.

3. Entropy-based regularization of adaboost

Entropy (information entropy, Shannon entropy) can be understood as a measure of information present in data produced by a probabilistic source. Given a discrete random variable x with values $\{x_1, x_2, \ldots, x_k\}$ with the probability function $p(x)$, the entropy can be given as

$$\text{Entropy}(x) = -\sum_{i=1}^{k} p(x_i) \log_b p(x_i). \tag{8}$$

If b is 2, the unit of entropy is a bit. In the case of $p(x_i) = 0$ for some x_i, the term $p(x_i) \log_b p(x_i)$ is taken as 0. It can be easily observed that entropy has its maximum value for a uniform distribution of x, i.e., if $p(x_i) = 1/N$.

The motivation to use the entropy as the regularization term comes from the idea of treating the features selected by the weak learners as a random variable, with the probability equal to the selection rate observed for the weak learners trained in the previous iterations. Instead of selecting the next weak learner as the one minimizing the weighted error, we additionally require that the entropy of the selection rate of the features selected be all weak learners is maximized. For example, if two decision stumps have already been trained, and both selected a given feature, say with index 4, the third trained weak learner might also want to select feature 4. However, it will be preferable to select other feature, for example with index 7, as this will increase the entropy. In the first case, we would have $\{p(x_4) = 1, p(x_i) = 0, \text{for } i \neq 4\}$ and in the second instance $\{p(x_4) = 2/3, p(x_7) = 1/3, p(x_i) = 0, \text{for } i \notin \{4, 7\}\}$. The entropy is higher in the second case, and in effect, the algorithm is forced to focus also on other features. Thus, the number of weak learners using the given feature (and also the total sum of weights α_m associated with a given feature) is limited. This is a mechanism similar to the forgetting effect that is used for NN training.

Obviously, the usage of the entropy is conflicting with the main criterion of minimizing the weighted error. The resulting boosting algorithm cannot be expected to select the features completely uniformly. In each step of the proposed algorithm, a weak learner is selected which minimizes the new criterion – the weighted error combined with the normalized entropy scaled with a parameter η. The entropy term will not make the boosting choose the features uniformly. Its influence is more subtle, as it will be presented in the section with numerical experiments.

The detailed algorithm is given as Algorithm 3. The change in the algorithm is primarily concerned with the training of the weak learner. Thus, we present the modified algorithm for decision stump selection. The whole framework of AdaBoost remains unchanged, only in step 2a of Algorithm 1 it is assumed that the list of the features selected by all decision stumps from the previous iterations are sent as the list F to Algorithm 3. In Algorithm 3, the entropy of the current selection rate of features has to be calculated (Eq. (11)). As the original criterion (weighted error, Eq. (6)) is to be minimized, and we prefer the entropy to be maximized, we actually use the negative entropy to be consistent with the minimization formulation of the main selection criterion. We normalize the entropy value by dividing it by the number of features in order to more easily adjust to classification problems with different numbers of features. Finally, the parameter η allows us to adjust the influence of the entropy term on the learning process (Eq. (9)). This is a new meta-parameter of the proposed algorithm, and, in practice, it has to be adjusted empirically by a method such as cross-validation.

Input: Training data $(\mathbf{x}_i, y_i)$, $y_i \in \{-1, 1\}$, $x_i \in \mathbb{R}^K$, K – the number of features, $i = 1, ..., N$;
$\{w_i\}$ – the weight distribution of the training examples;
F – the list of features selected by the previously trained decision stumps;
η – the parameter to tune the importance of the entropy term.
Output: A decision stump $h(\mathbf{x})$ minimazing the weighted classification error regularized by the entropy term; $h(\mathbf{x}) = \{j, threshold, class\_id\}$, where $class\_id \in \{-1, 1\}$, $j \in \{1, ..., K\}$, $threshold \in \mathbb{R}$.
1. Find j, $threshold$, and $class\_id$, so that for $h(\mathbf{x}) = \{j, threshold, class\_id\}$, the following criterion is minimized

$$error'(h(\mathbf{x})) = error(h(\mathbf{x})) - \eta \frac{Entropy(F \cup \{j\})}{K} \tag{9}$$

where

$$error(h(\mathbf{x})) = \sum_{i=1}^{N} w_i I(h(\mathbf{x}_i \neq y_i)) \tag{10}$$

and

$$Entropy(F \cup \{j\}) = -\sum_{l=1}^{K} P_l log(P_l) \tag{11}$$

and P_l is the probability of finding the feature l on the list $F \cup \{j\}$.
2. **Return** $h(\mathbf{x}) = \{j, threshold, class\_id\}$ as

$$h(\mathbf{x}) = \begin{cases} class\_id & \text{if } x_j \geq threshold \\ -1 \cdot class\_id & \text{if } x_j < threshold \end{cases} \tag{12}$$

Algorithm 3: Decision stump learning in EntropyAdaBoost.

4. Numerical experiments

Eleven classification problems from the UCI Repository [14] were collected to test the proposed entropy-based regularization of AdaBoost. They are all two-class classification problems with a varying number of features and available examples. The databases are summarized in Table 1. Ten-fold cross-validation was used as the testing procedure. The compared algorithms were AdaBoost as in Algorithm 1, ϵ-AdaBoost, and the proposed EntropyAdaBoost as in Algorithm 3. Each algorithm was allowed to create a maximum number of 500 weak learners as decision stumps. We reported the best performing aggregated classifier, i.e., the number of the weak learners with the best mean test error in each problem. The results are presented in Table 2, where the mean test error rate (*t.e.r.*), standard deviation (*std*) and the best performing number of decision stumps are given for each algorithm and problem. Additionally, for ϵ-AdaBoost the best performing value of ϵ and for EntropyAdaBoost the best performing value of η are given. The parameters ϵ and η are meta-parameters that had to be tuned for each problem separately. Different values were tried. In each

Table 1. Summary of the databases used in the numerical experiments.

Database	Features	Examples
Pima	8	768
Parkinsons	22	195
Sonar	60	208
Ionosphere	33	351
QSAR	41	1055
ILPD	10	579
SPECTF	44	267
Messidor	19	1151
Wisconsin diagnostic	30	569
Wisconsin original	9	683
Wisconsin prognostic	32	194

Table 2. Results of the ten-fold cross-validation tests.

Dataset	AdaBoost			ϵ-AdaBoost				EntropyAdaBoost			
	t.e.r.	std	stumps	t.e.r.	std	stumps	ϵ	t.e.r.	std	stumps	η
Pima	0.233	0.055	43	0.225	0.050	148	0.01	0.230	0.051	43	0.001
Parkinsons	0.058	0.053	364	0.046	0.052	495	1.0	0.053	0.050	168	0.5
Sonar	0.143	0.080	415	0.135	0.072	50	0.1	0.134	0.072	426	0.05
Ionosphere	0.080	0.054	92	0.068	0.044	128	0.1	0.076	0.033	52	0.05
QSAR	0.137	0.036	171	0.132	0.024	130	0.1	0.133	0.029	280	0.05
ILPD	0.316	0.060	430	0.309	0.060	353	0.1	0.307	0.049	387	0.5
SPECTF	0.188	0.066	89	0.183	0.073	49	0.1	0.188	0.066	89	1E-10
Messidor	0.276	0.030	255	0.279	0.047	452	0.1	0.275	0.030	255	0.0005
Wisconsin diagnostic	0.015	0.015	64	0.016	0.012	213	1.0	0.015	0.015	64	1E-10
Wisconsin original	0.034	0.019	28	0.026	0.019	32	0.1	0.034	0.019	27	0.5
Wisconsin prognostic	0.320	0.096	422	0.313	0.094	47	0.1	0.297	0.062	438	0.1

problem, ϵ-AdaBoost was run with ϵ values equal to $1.0 * k$ and $0.5 * k$ for $k = 1, 1e-1, \ldots, 1e-5$. In each problem, η values tested were $1.0 * k$ and $0.5 * k$ for $k = 1, 1e-1, \ldots, 1e-10$. The best performing values are reported in Table 2 in the respective columns. In columns *t.e.r.* (test error rate) underlined are the values showing lower test errors for ϵ-AdaBoost and EntropyAdaBoost compared to AdaBoost. Similarly, in columns *stumps*, underlined are the values indicating a lower or equal number of weak learners compared to AdaBoost.

It can be observed from Table 2, that neither ϵ-AdaBoost nor EntropyAdaBoost can improve the results of AdaBoost in each case. Let us first compare the mean t.e.rs. Improvement was possible in almost all cases. EntropyAdaBoost never performed worse than AdaBoost. It improved the test errors for eight problems, while for the remaining three the test errors were the same. ϵ-AdaBoost improved the results of AdaBoost in nine problems. On the other hand, it produced worse test errors in two cases (Messidor and Wisconsin diagnostic).

To compare the results in a more rigorous way, we used a statistical test to compare the results. As suggested in [15, 16], a proper way to compare the results of several (more than two) algorithms on several problems (the number of problems should be bigger than the number of the algorithms), a Friedman test should be used. The null hypothesis of the Friedman test states that there is no difference in the general performance of all algorithms considering all of the test problems at once. This approach allows comparing the algorithms in general, and not with respect to only one particular classification problem. In the case when the null hypothesis is rejected (which happens when $p - value$ is sufficiently small, usually smaller than 0.05), proper pair-wise post-hoc comparisons are performed to discover the individual differences among the algorithms. This two-step testing procedure is considered a safer way to carry out statistical tests than performing only many pair-wise comparisons (such as many pairwise Wilcoxon tests). This is due to the fact that in the latter case there is a bigger probability of making at least one type I error, i.e., to incorrectly reject the null hypothesis, which in our case would be signaling the differences between algorithms, which in fact are performing equally well in general. More details about the adopted testing procedure (Friedman test and Shaffer post-hoc procedures) can be found in [15, 16].

The Friedman test and the Shaffer post-hoc comparisons were performed using the Keel software package [17, 18]. In Table 3, the ranks assigned to the algorithms by the Friedman test are presented. ϵ-AdaBoost and EntropyAdaBoost are ranked higher (with similar ranks) than AdaBoost. $P - value$ computed by the Friedman test was 0.0127, which is smaller than 0.05 and suggests that the described above null hypothesis can be rejected. This allowed running pair-wise comparisons as the Shaffer post-hoc procedures. The adjusted $p - values$ from the Shaffer post hoc tests are presented in Table 4. They suggest that on the level of significance of 0.05, both ϵ-AdaBoost and

Table 3. Rankings of the algorithms by Friedman test ($p - value = 0.0127$).

Algorithm	Ranking
EntropyAdaBoost	1.6364
ϵ-AdaBoost	1.6364
AdaBoost	2.7273

Table 4. Adjusted $p - values$ from Shaffer post-hoc pairwise comparisons.

Hypothesis	p_{Shaffer}
AdaBoost vs. ϵ-AdaBoost	0.031546
AdaBoost vs. EntropyAdaBoost	0.031546
ϵ-AdaBoost vs. EntropyAdaBoost	1

EntropyAdaBoost perform in general better than AdaBoost ($p - values < 0.05$). On the other hand, there is no significant difference between ϵ-AdaBoost and EntropyAdaBoost ($p - value = 1 > 0.05$).

Considering only the classification accuracy, both regularization algorithms, ϵ-AdaBoost, and EntropyAdaBoost can be used, as in general, they perform better than AdaBoost. On the other hand, after analyzing the obtained results in detail, it is recommended to try both of them, as in a given classification problem one of them might be preferable. This is even more evident if we consider also a number of the weak classifiers that were needed by a given algorithm to solve the classification problem. In many problems, we want to have the number of the weak learners as small as possible, to provide simpler and faster final classifier. In Table 5, for each problem and for each regularization algorithm, we mark x for the cases in which the algorithm provided smaller test error with a smaller or equal number of weak learners, or the same test error with a strictly smaller number of weak learners. By analyzing the results in Table 5, it can be seen that ϵ-AdaBoost and EntropyAdaBoost are highly complementary. If we consider both criteria (the test error and the number of weak learners) we can see that neither ϵ-AdaBoost nor EntropyAdaBoost always improves over AdaBoost. There are cases when the test error is smaller, but a bigger number of weak learners is needed. If we care about improving both, the test error and simplicity of the classifier, it is definitely not enough to use only ϵ-AdaBoost, and the proposed EntropyAdaBoost is a useful tool to complement ϵ-AdaBoost. The total number of stumps for all problems for AdaBoost was 2373. Selecting the best option based on Table 5 decreases this number to 1238, which is 52.2% of the original number. Thus, the possible reduction is considerable.

Table 5. Marked results with both criteria (the test error and the number of the weak learners) improved.

Dataset	Improvement compared to AdaBoost	
	ϵ-AdaBoost	EntropyAdaBoost
Pima		x
Parkinsons		x
Sonar	x	
Ionosphere		x
QSAR	x	
ILPD	x	x
SPECTF	x	
Messidor		x
Wisconsin diagnostic		
Wisconsin original		x
Wisconsin prognostic	x	

Figure 1 shows the example dependence of the mean test error rate on the number of weak learners (for Parkinson's database). The proposed EntropyAdaBoost algorithm is able to use new base classifiers more efficiently and reach lower error faster than the others. To explain the mechanism of EntropyAdaBoost, Fig. 2 shows the selection rates of features for the three algorithms from all of the 10 runs of the cross-validation for ILPD database. It can be seen that both regularization algorithms change the selection rate of the features, but they are doing it in different ways. Including the entropy term in EntropyAdaBoost does not lead to uniform selection rates as the influence of this factor is limited by the value of the parameter η. However, the selection rates are altered in a subtle way, different from that in ϵ-AdaBoost. We can define an accumulated weight of a given feature as the sum of the weights (values of α_m) of all decision stumps that selected this feature. The comparison of the accumulated weights of the features is presented in Fig. 3 for ILPD database.

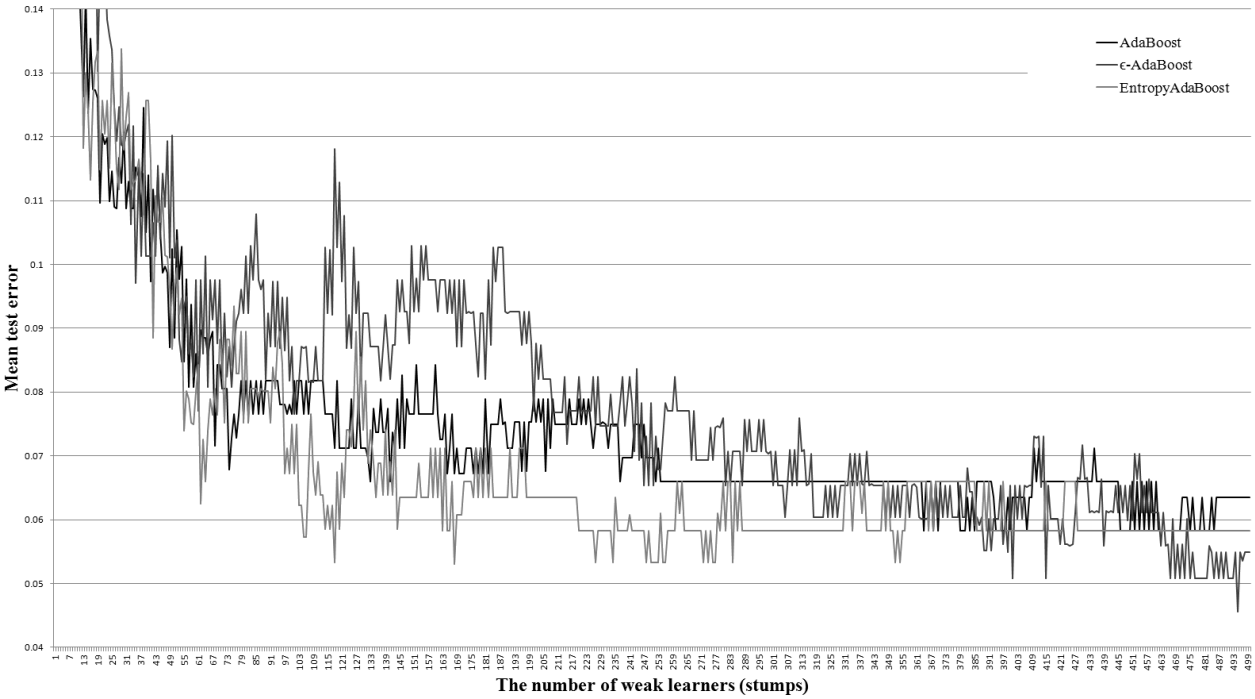

Fig. 1. Dependance of the mean test error on the number of weak learners (Parkinsons dataset).

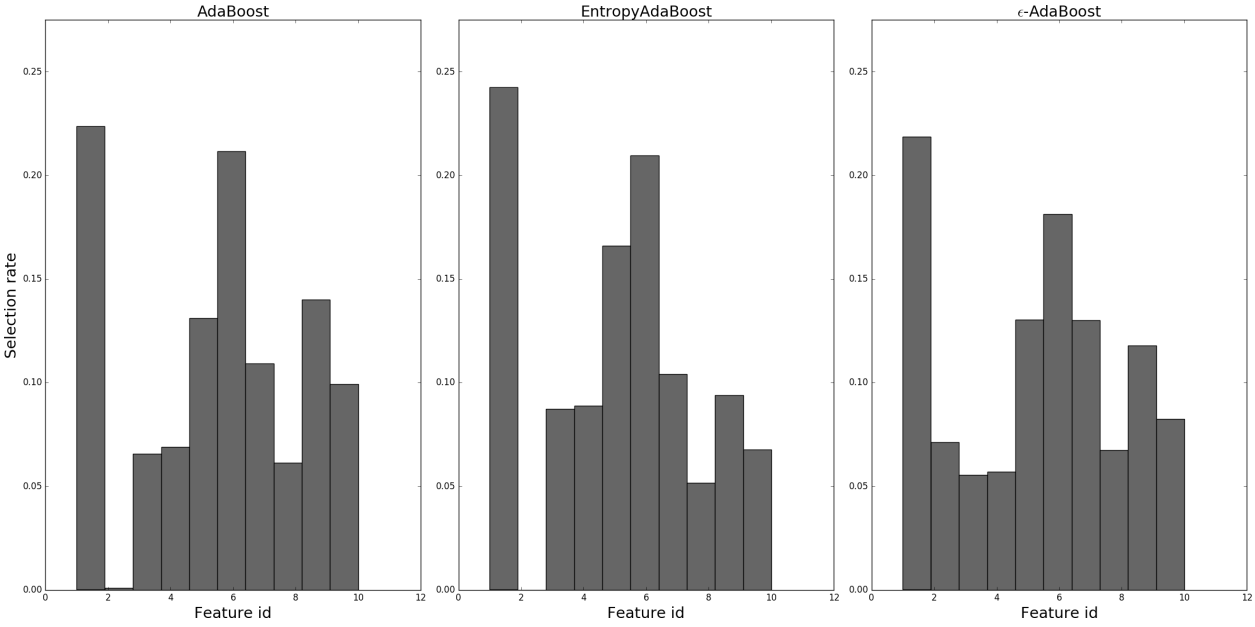

Fig. 2. Features' selection rates (ILPD database).

Additionally it shows that EntropyAdaBoost changes also these values. It is interesting that it can lead to smaller values than in the case of AdaBoost or ϵ-AdaBoost, which can be a positive outcome as this is in the case of the connection weights of neurons in NN.

Including the entropy term in the proposed method influences the boosting procedure in a subtle way, and the results do not have to look always as in the example figures. However, given the overall performance, it is beneficial for the boosting procedure.

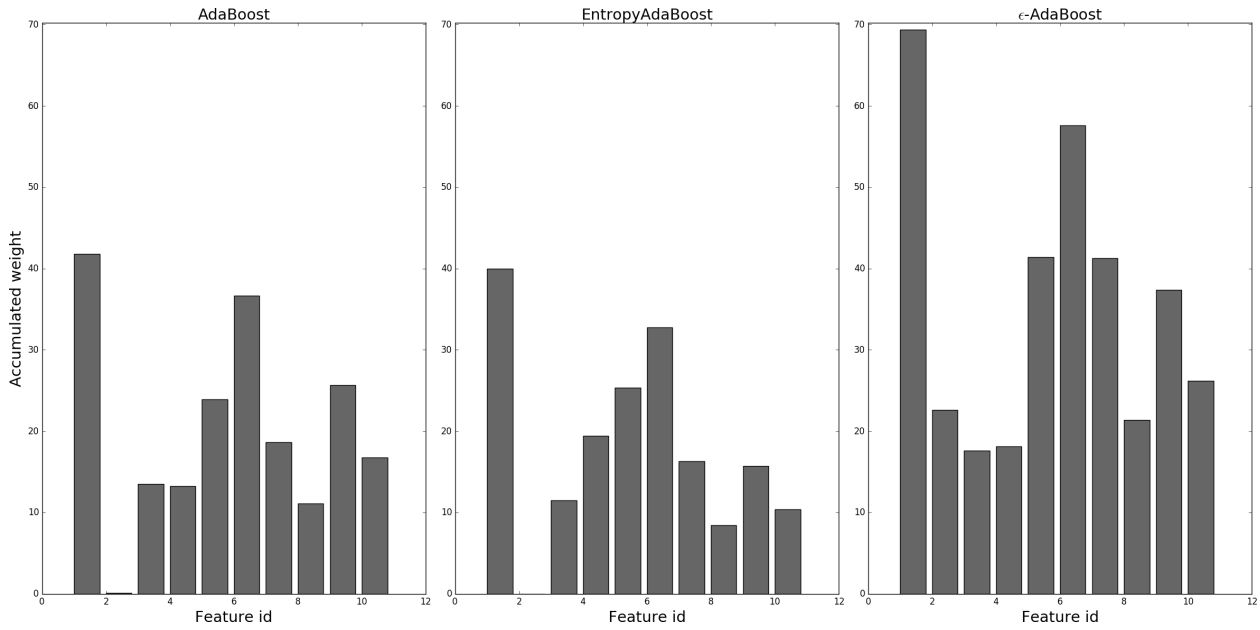

Fig. 3. Features' accumulated weights (ILPD database).

5. Conclusions

In this paper, we proposed a novel method to regularize the boosting procedure of AdaBoost algorithm. With the aim to prevent the algorithm to focus only on a small subset of features and encourage it to use as many features as possible, an additional term based on entropy was added to the principal criterion used to select the weak learner in each iteration of boosting. This additional term measured the entropy of the selection rates of features chosen by the trained weak learners. This entropy was normalized by the number of available features to adapt the algorithm to the problems with different numbers of features. The influence of the entropy term was limited by parameter η. We compared the proposed EntropyAdaBoost algorithm with the original AdaBoost and its regularized version, ϵ-AdaBoost. The results showed a general improvement over AdaBoost when considering the test errors. When considering the test error and the number of weak learners (which we want to minimize to provide simpler and faster final classifiers), the proposed method and the ϵ-AdaBoost are highly complementary. In all classification problems tested, with the exception of one, it was always possible to improve AdaBoost considering both criteria. This shows that the proposed EntropyAdaBoost and ϵ-AdaBoost used together constitute a strong tool to tune the boosting classifiers, when both the test error and the number of weak learners have to be minimized.

References

[1] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd edition, Springer Series in Statistics, 2009. http://www.springer.com/gp/book/9780387848570.

[2] R. Meir, G. Rätsch. An introduction to boosting and leveraging. In: S. Mendelson, A.J. Smola [Eds.], *Advanced Lectures on Machine Learning, Summer School 2002 Canberra, Australia, February 11–22, 2002 Revised Lectures*, 118–183, Springer-Verlag New York, Inc., NY, USA, 2003. https://link.springer.com/chapter/10.1007%2F3-540-36434-X_4.

[3] R.E. Schapire. Theoretical views of boosting and applications. In: O. Watanabe, T. Yokomori [Eds.], *Algorithmic Learning Theory: 10th International Conference, ALT99 Tokyo, Japan, December 6–8, 1999 Proceedings*, pp. 13–25, Springer, Berlin/Heidelberg, 1999. https://link.springer.com/chapter/10.1007/3-540-46769-6_2.

[4] P. Viola, M.J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, **57**(2): 137–154, 2004. https://link.springer.com/article/10.1023/B:VISI.0000013087.49260.fb

[5] W. Jiang. Is regularization unnecessary for boosting? In: *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics* (AISTATS), 2001. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.5229.

[6] Y. Xi, Z. Xiang, P. Ramadge, R. Schapire. Speed and sparsity of regularized boosting. In: D. van Dyk, M. Welling [Eds.], *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, Clearwater Beach, Florida, USA, Vol. 5 of JMLR: W&CP 5, pp. 615–622, 2009. http://proceedings.mlr.press/v5/xi09a.html.

[7] P. Bühlmann, T. Hothorn. Boosting algorithms: regularization, prediction and model fitting, *Statistical Science*, **22**: 477–505, 2007. https://www.jstor.org/stable/27645854.

[8] C. Shen, H. Li, A. van den Hengel. Fully corrective boosting with arbitrary loss and regularization. *Neural Networks*, **48**: 44–58, 2013. http://www.sciencedirect.com/science/article/pii/S0893608013001913.

[9] M.K. Warmuth, J. Liao, G. Rätsch. Totally corrective boosting algorithms that maximize the margin. In: *Proceedings of the 23rd International Conference on Machine Learning*, ACM, New York, NY, USA, pp. 1001–1008, 2006. https://users.soe.ucsc.edu/~manfred/pubs/C75.pdf.

[10] D.D. Le, S. Satoh. Ent-Boost: boosting using entropy measures for robust object detection. *Pattern Recognition Letters*, **28**: 1083–1090, 2007. http://www.sciencedirect.com/science/article/pii/S0167865507000190.

[11] R.E. Schapire. A brief introduction to boosting. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (IJCAI), **2**: 1401–1406, 1999. https://www.cs.utah.edu/~piyush/teaching/brief_intro_boosting.pdf.

[12] R.E. Schapire, Y. Freund. *Boosting: Foundations and Algorithms*. The MIT Press, 2012. https://mitpress.mit.edu/books/boosting.

[13] S. Rosset, J. Zhu, T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, **5**: 941–973, 2004. http://www.jmlr.org/papers/volume5/rosset04a/rosset04a.pdf.

[14] M. Lichman. *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science, 2013. http://archive.ics.uci.edu/ml.

[15] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, **7**: 1–30, 2006. http://www.jmlr.org/papers/v7/demsar06a.html.

[16] J. Derrac, S. García, D. Molina, F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, **1**: 3–18, 2011. http://www.sciencedirect.com/science/article/pii/S2210650211000034.

[17] J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, F. Herrera. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, **13**: 307–318, 2008. https://link.springer.com/article/10.1007/s00500-008-0323-y.

[18] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, **17**: 255–287, 2011. http://sci2s.ugr.es/keel/pdf/keel/articulo/2011-KEEL-dataset-MVLSC.pdf.